DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

PMC BiSerial III ORB2 Base & Channels

Driver Documentation

Win32 Driver Model

Manual Revision B
Corresponding Hardware: Revision A
10-2005-0204
Corresponding Firmware:
ORB2: Design 9, Revision 3

ORB2Base & ORB2Chan
WDM Device Drivers for the
PMC-BiSerial-III-ORB2
8 port Multi-function IO

Dynamic Engineering
150 DuBois St., Suite C
Santa Cruz, CA 95060
(831) 457-8891
FAX: (831) 457-4793

Manual Revision B Revised March 12, 2009

Table of Contents

Introduction 5
Note 6
Driver Installation 7
Windows 2000 Installation 8
Windows XP Installation 8
Driver Startup 9
IOCTL_ORB2_BASE_GET_INFO 10
IOCTL_ORB2_BASE_LOAD_PLL_DATA 10
IOCTL_ORB2_BASE_READ_PLL_DATA 11
IOCTL_ORB2_BASE_SET_BASEREG 11
IOCTL_ORB2_BASE_GET_BASEREG 11
IOCTL_ORB2_BASE_GET_STATUS 11
IOCTL_ORB2_BASE_SET_GPIOTERM 12
IOCTL_ORB2_BASE_GET_GPIOTERM 12
IOCTL_ORB2_BASE_SET_GPIODIR 12
IOCTL_ORB2_BASE_GET_GPIODIR 12
IOCTL_ORB2_BASE_SET_GPIOREG 12
IOCTL_ORB2_BASE_GET_GPIOREG 12
IOCTL_ORB2_BASE_GET_GPIOIN 13
IOCTL_ORB2_CHAN_GET_INFO 14
IOCTL_ORB2_CHAN_GET_STATUS 14
IOCTL_ORB2_CHAN_CLR_STATUS 15
IOCTL_ORB2_CHAN_SET_FIFO_LEVELS 15
IOCTL_ORB2_CHAN_GET_FIFO_LEVELS 15
IOCTL_ORB2_CHAN_GET_FIFO_COUNTS 15
IOCTL_ORB2_CHAN_RESET_FIFOS 16
IOCTL_ORB2_CHAN_REGISTER_EVENT 16
IOCTL_ORB2_CHAN_ENABLE_INTERRUPT 16
IOCTL_ORB2_CHAN_DISABLE_INTERRUPT 16
IOCTL_ORB2_CHAN_FORCE_INTERRUPT 16
IOCTL_ORB2_CHAN_GET_ISR_STATUS 17
IOCTL_ORB2_CHAN_SWW_TX_FIFO 17
IOCTL_ORB2_CHAN_SWR_RX_FIFO 17
IOCTL_ORB2_CHAN_SET_CONT 17
IOCTL_ORB2_CHAN_GET_CONT 17
IOCTL_ORB2_CHAN_SET_TX_REG 18
IOCTL_ORB2_CHAN_GET_TX_REG 18

IOCTL_ORB2_CHAN_SET_TX_COUNT 18
IOCTL_ORB2_CHAN_GET_TX_COUNT 18
Ternary Control 19
IOCTL_ORB2_CHAN_SET_TX_COM12 19
IOCTL_ORB2_CHAN_GET_TX_COM12 19
IOCTL_ORB2_CHAN_SET_RX_COM12 19
IOCTL_ORB2_CHAN_GET_RX_COM12 19
IOCTL_ORB2_CHAN_SET_SYNC_COM12 19
IOCTL_ORB2_CHAN_GET_SYNC_COM12 19
LS Control 20
IOCTL_ORB2_CHAN_SET_TX_COM34 20
IOCTL_ORB2_CHAN_GET_TX_COM34 20
IOCTL_ORB2_CHAN_SET_RX_COM34 20
IOCTL_ORB2_CHAN_GET_RX_COM34 20
TLM Control 21
IOCTL_ORB2_CHAN_SET_MAS_COM56 21
IOCTL_ORB2_CHAN_GET_MAS_COM56 21
IOCTL_ORB2_CHAN_SET_TAR_COM56 21
IOCTL_ORB2_CHAN_GET_TAR_COM56 21
TLM Register Files 21
IOCTL_ORB2_CHAN_SET_MAS_RF_COM56 21
IOCTL_ORB2_CHAN_GET_MAS_RF_COM56 21
IOCTL_ORB2_CHAN_SET_TAR_RF_COM56 21
IOCTL_ORB2_CHAN_GET_TAR_RF_COM56 22
HS Control 22
IOCTL_ORB2_CHAN_SET_TX_COM78 22
IOCTL_ORB2_CHAN_GET_TX_COM78 22
IOCTL_ORB2_CHAN_SET_RX_COM78 22
IOCTL_ORB2_CHAN_GET_RX_COM78 22
Length Error Counts 23
IOCTL_ORB2_CHAN_SET_RX_LEN_ERR_CNT 23
IOCTL_ORB2_CHAN_GET_RX_LEN_ERR_CNT 23
Write 24
Read 24
Warranty and Repair 25
Service Policy 26
Out of Warranty Repairs 26
For Service Contact: 26
Appendix 27
Reference copy of structures for evaluation 27
Base: 27
Channel: 28

Introduction

The ORB2Base and ORB2Chan drivers are Win32 driver model (WDM) device drivers for the PMC-BiSerial-III-ORB2 from Dynamic Engineering.

The ORB2 driver package has three parts. The driver is installed into the Windows® OS, the test executable and the User Application "Userap" executable.

The driver and test are delivered as installed or executable items to be used directly or indirectly by the user. The Userap code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

The "test" executable allows the user to use the driver in script form from a DOS window. Each driver call can be accessed, parameters set and returned. Normally not needed or used by the integrator, but a very handy tool in certain circumstances. The test executable has a "help" menu to explain the calls, parameters and returned information.

UserAp is a stand-alone code set with a simple, and powerful menu plus a series of "tests" that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering. For example most Dynamic Engineering PCI based designs support DMA. DMA is demonstrated with the memory based loop-back tests. The tests can be ported and modified to fit your requirements.

The test software can be ported to your application to provide a running start. It is recommended to port the switch and status tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system.

The hardware has features common to the board level and features that are set apart in "channels". The channels have the same offsets within the channel, and the same status and control bit locations allowing for symmetrical software in the calling routines. The driver supports the channels with a variable passed in to identify which channel is being accessed. The hardware manual defines the pinout for each channel and the

bitmaps and detailed configurations for each channel. The driver handles all aspects of interacting with the channels and base features.

We strive to make a useable product, and while we can guarantee operation we can't foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

The PMC BiSerial III board has a Spartan3-4000 Xilinx FPGA to implement the PCI interface, FIFOs and protocol control and status for the IO. The IO are grouped into COM ports. COM1&2 are defined to be Ternary, COM3&4 are LowSpeed, COM5&6 are Telemetry, and COM7&8 are HighSpeed ports. Channel A of the PLL is set to 140 MHz and then divided down to control COM1-6. COM7&8 use PLL channel B. Please refer to the HW manual for a much more complete description of the HW features.

When the PMC-BiSerial-III-ORB2 board is recognized by the PCI bus configuration utility it will start the PmcBis3ORB2Base driver which will create a device object for each board, initialize the hardware, create a child devices for the channel and request loading of the PmcBis3ORB2Chan driver. The PmcBis3ORB2Chan driver will create a device object for the I/O channel and perform initialization on the channel. IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of data in and out of the device.

Note

This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. For more detailed information on the hardware implementation, refer to the PMC-BiSerial-III-ORB2 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include driver: ORB2Base.sys, PmcORB2.inf, DDORB2Base.h, ORB2BaseGUID.h, ORB2Chan.sys, DDORB2Chan.h, ORB2ChanGUID.h. Driver Test: ORB2Test.exe, Userap: User Application source files.

ORB2BaseGUID.h and ORB2ChanGUID.h are C header files that define the device interface identifiers for the drivers. DDORB2Base.h and DDORB2Chan.h files are C header files that define the Application Program Interface (API) to the drivers. These files are required at compile time by any application that wishes to interface with the drivers, but they are not needed for driver installation. The files are included with the Userap fileset.

ORB2Test.exe is a sample Win32 console applications that makes calls into the ORB2Base/ORB2Chan drivers to test each driver call without actually writing any application code. They are not required during driver installation either. Please note that the test driver software is incomplete at this time. Please refer to the User Application software package as a reference for using the driver.

To run ORB2Test, open a command prompt console window and type ***ORB2Test -d0 -?*** to display a list of commands (the PmcBis3ORB2Test.exe file must be in the directory that the window is referencing). The commands are all of the form ***ORB2Test -dn -im*** where ***n*** and ***m*** are the device number and PmcBis3ORB2Base driver ioctl number respectively or ***ORB2Test -cn -im*** where ***n*** and ***m*** are the channel number (0-1) and PmcParTtlORB2Chan driver ioctl number respectively.

This test application is intended to test the proper functioning of each driver call, **not** for normal operation. Many integration efforts will never need the debugger capability that the test menu represents. The test capability will allow the designer to access the card without any other software in the way to make sure that the system can “see” the card and to do basic card manipulations.

Windows 2000 Installation

Copy PmcORB2.inf, ORB2Base.sys and ORB2Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Select ***Next***.
_ Select ***Search for a suitable driver for my device.***
_ Select ***Next***.
_ Insert the disk prepared above in the desired drive.
_ Select the appropriate drive e.g. ***Floppy disk drives***.
_ Select ***Next***.
_ The wizard should find the PmcORB2.inf file.
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Repeat this for each channel as necessary.

Windows XP Installation

Copy PmcORB2.inf, ORB2Base.sys and ORB2Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Insert the disk prepared above in the desired drive.
_ Select ***No when asked to connect to Windows Update***.
_ Select ***Next***.
_ Select ***Install the software automatically.***
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Proceed as above for each channel as necessary.

Driver Startup

Once the drivers have been installed they will start automatically when the system recognizes the hardware.

Handles can be opened to a specific board by using the CreateFile() function call and passing in the device names obtained from the system.

The interfaces to the devices are identified using globally unique identifiers (GUIDs), which are defined in ORB2BaseGUID.h and ORB2ChanGUID.h.

The User Application software contains a file called “main.c”. Main has the initialization needed to get the handles to the base and channel assets of the installed PMC-BiSerial-III-ORB2 device.

The main file provided is designed to work with our test menu and includes user interaction steps to allow the user to select which board is being tested in a multiple board environment. The integrator can hardcode for single board systems or use an automatic loop to operate in multiple board systems without using user interaction. For multiple user systems it is suggested that the board number is associated with a switch setting so the calls can be associated with a particular board from a physical point of view.

IO Controls

The drivers use IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board or I/O channel. IOCTLs are called using the Win32 function DeviceIoControl() (see below), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with
CreateFile()
  DWORD         dwIoControlCode, // Control code defined in API
header file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned, // Pointer to return length
parameter
  LPOVERLAPPED lpOverlapped,     // Optional pointer to
overlapped structure
);                                 // used for asynchronous I/O
```

The IOCTLs defined for the ORB2Base driver are described below:

IOCTL_ORB2_BASE_GET_INFO

Function: Return the Instance Number, Switch value, PLL device ID, Xilinx rev and Current Driver Version

Input: None

Output: ORB2_BASE_DRIVER_DEVICE_INFO : Structure

Notes: Switch value is the configuration of the on-board dip-switch that has been set by the User (see the board silk screen for bit position and polarity). The PLL ID is the device address of the PLL device. This value, which is set at the factory, is usually 0x69 but may also be 0x6A. See DDORB2Base.h for the definition of SPWR_BASE_DRIVER_DEVICE_INFO.

IOCTL_ORB2_BASE_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.

Input: ORB2_BASE_PLL_DATA structure

Output: None

Notes:

IOCTL_ORB2_BASE_READ_PLL_DATA
Function: Returns the contents of the PLL's internal registers
Input: None
Output: ORB2_BASE_PLL_DATA structure
Notes: The register data is output in the ORB2_BASE_PLL_DATA structure in an array of 40 bytes.

IOCTL_ORB2_BASE_SET_BASEREG
Function: Write to Base Control Register - general access to base control register of card, use with bit definitions
Input: ULONG
Output: none
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_ORB2_BASE_GET_BASEREG
Function: Read from Base Control Register - general access from base control register of card, use with bit definitions
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_ORB2_BASE_GET_STATUS
Function: Read from Status Register
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access from the register. See DDORB2Base.h for bit map information. See the HW manual for exact definitions of bits.

There is an 8 bit parallel port for General Purpose IO [GPIO]. The following calls are used to set direction, termination, and data values for each bit.

IOCTL_ORB2_BASE_SET_GPIOTERM
Function: Write to GPIO Termination Register
Input: ULONG 7-0 correspond to GPIO7-0
Output: none
Notes: 0 = not terminated, 1 = terminated for each bit.

IOCTL_ORB2_BASE_GET_GPIOTERM
Function: Read from Direction Register
Input: none
Output: ULONG 7-0 valid
Notes: 0 = not terminated, 1 = terminated for each bit.

IOCTL_ORB2_BASE_SET_GPIODIR
Function: Write to GPIO Direction Register
Input: ULONG 7-0 correspond to GPIO7-0
Output: none
Notes: 0 = Rx, 1 = Tx for each bit.

IOCTL_ORB2_BASE_GET_GPIODIR
Function: Read from Direction Register 7-0
Input: none
Output: ULONG 7-0 valid
Notes: 0 = Rx, 1 = Tx for each bit.

IOCTL_ORB2_BASE_SET_GPIOREG
Function: Write to GPIO Data Register
Input: ULONG 7-0 correspond to GPIO7-0
Output: none
Notes: 0 or 1 for each bit. The bits marked for transmit in the direction register are actually driven out. The other bits remain in the register but are not driven to the IO.

IOCTL_ORB2_BASE_GET_GPIOREG
Function: Read from Direction Register
Input: none
Output: ULONG
Notes:

IOCTL_ORB2_BASE_GET_GPIOIN
Function: Read from IO Register 7-0
Input: none
Output: ULONG 7-0 correspond to GPIO 7-0
Notes: data from IO lines. May not match data register if some bits are masked with the direction register.

The IOCTLs defined for the PmcBis3ORB2Chan driver are described below:
There are 8 channels. Each channel corresponds to a COM port. The COM ports are numbered 1-8 and the channels 0-7. COM1 = channel 0 … COM8 = channel 7. COM1&2 support Ternary. COM 3&4 support LS. COM5&6 support TLM. COM7&8 support HS. It is important to use the calls associated with the channel being used or unexpected results will occur. There is a great deal of overlap so you may get away with it in some cases. The TX and RX control registers have separate calls using separate structures to help organize your software and eliminate mistakes. Calls that can be used across all of the channels are not numbered. Calls that only work with specific channel pairs are numbered based on the COM ports supported – COM12 for example.

IOCTL_ORB2_CHAN_GET_INFO
Function: Return the Instance Number and Current Driver Version
Input: None
Output: ORB2_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of ORB2_CHAN_DRIVER_DEVICE_INFO in the DDORB2Chan.h header file.

IOCTL_ORB2_CHAN_GET_STATUS
Function: Return the value of the status register and clear latched bits
Input: None
Output: Status register value(ULONG)
Notes: Latched interrupt status and DMA error bits are cleared by read – [call writes back and clears bits]. Other Latched Error bits not cleared by read. See quick reference status bits below. Defines available in DDORB2Chan.h Detailed definitions are available in the HW manual.

```
STAT_TX_FIFO_MT       0x00000001 //0 set when TX FIFO is empty
STAT_TX_FIFO_AE       0x00000002 //1 set when TX FIFO is Almost Empty
STAT_TX_FIFO_FULL     0x00000004 //2 set when TX FIFO is Full
STAT_RX_FIFO_MT       0x00000010 //4 set when RX FIFO is Empty
STAT_RX_FIFO_AF       0x00000020 //5 set when RX FIFO is Almost Full
STAT_RX_FIFO_FULL     0x00000040 //6 set when RX FIFO is Full
STAT_TX_DONE_INT      0x00000100 //8 Transmit Done Interrupt Occurred
STAT_RX_DONE_INT      0x00000200 //9 Receive Done Interrupt Occurred
STAT_TX_FIFO_INT      0x00000400 //10 Transmit FIFO Interrupt Occurred
STAT_RX_FIFO_INT      0x00000800 //11 Receive FIFO Interrupt Occurred
STAT_WR_DMA_ERR       0x00001000 //12 write DMA error
STAT_RD_DMA_ERR       0x00002000 //13 read DMA error
STAT_WR_DMA_INT       0x00004000 //14 write DMA Interrupt
STAT_RD_DMA_INT       0x00008000 //15 read DMA Interrupt
STAT_VERBOSE_OVFL 0x00020000 // 17 FIFO Full when time to write in Verbose mode
STAT_RX_OVFL          0x00040000 //18 FIFO Full when time to write RX
```

STAT_TX_UNFL	0x00080000 //19 FIFO MT when time to read TX
STAT_RX_IDLE	0x00100000 //20 set when state-machine is in the idle state
STAT_TX_IDLE	0x00200000 //21 set when state-machine is in the idle state
STAT_DMA_RD_IDLE	0x00400000 //22 set when Burst Out [read] DMA state-machine is in the idle state
STAT_DMA_WR_IDLE	0x00800000 //23 set when Burst In [write] DMA state-machine is in the idle state
STAT_VERBOSE_IDLE	0x01000000 //24 set when Verbose SM is IDLE COM1/2 only
STAT_FIFO_MT	0x01000000 //24 set when External FIFO is MT COM7/8 only
STAT_FIFO_AMT	0x02000000 //25 set when External FIFO is Almost MT
STAT_FIFO_AFL	0x04000000 //26 set when External FIFO is Almost Full
STAT_FIFO_FL	0x08000000 //27 set when External FIFO is Full
STAT_DIR	0x10000000 //28 Set when direction is transmit
STAT_TX_IDLE	0x20000000 //29 spare
STAT_LOC_INT	0x40000000 //30 set when local interrupt is potentially active [not DMA], before mask
STAT_ACTIVE_INT	0x80000000 //31 channel interrupt is active [after mask and includes DMA]

IOCTL_ORB2_CHAN_CLR_STATUS
Function: Clear Error Bits latched and not cleared by status read
Input: ULONG
Output: none
Notes: Clear latched error bits. Allows polling on FIFO status without losing potential Error conditions. Write back with same bit position set to clear. Defines available in DDORB2Chan.h Detailed definitions are available in the HW manual.

IOCTL_ORB2_CHAN_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels for the channel.
Input: ORB2_CHAN_FIFO_LEVELS structure
Output: None
Notes: The FIFO counts are compared to these levels to determine the value of the STAT_TX_FF_AMT and STAT_RX_FF_AFL status bits.

IOCTL_ORB2_CHAN_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: ORB2_CHAN_FIFO_LEVELS structure
Notes:

IOCTL_ORB2_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in FIFO's.
Input: None
Output: ORB2_CHAN_FIFO_COUNTS structure

Notes: Returns the actual TX FIFO data counts and count including DMA pipeline RX FIFO.

IOCTL_ORB2_CHAN_RESET_FIFOS
Function: Resets one or both internal FIFOs for the referenced channel.
Input: ORB2_FIFO_SEL enumeration type See structure definition in DDORB2Chan.h
Output: None
Notes: Resets RX, TX, Both (TX and RX) , All FIFO's . ALL only applies to COM7&8.

IOCTL_ORB2_CHAN_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause the event to be signaled.

IOCTL_ORB2_CHAN_ENABLE_INTERRUPT

Function: Enables the channel Master Interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to user interrupts. The master interrupt enable is disabled in the driver interrupt service routine when a user interrupt is serviced. Therefore this command must be run after each interrupt occurs to re-enable it.

IOCTL_ORB2_CHAN_DISABLE_INTERRUPT

Function: Disables the channel Master Interrupt.
Input: None
Output: None
Notes: This call is used when user interrupt processing is no longer desired.

IOCTL_ORB2_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the channel master interrupt is enabled. This IOCTL is used for development, to test interrupt processing. Board level master interrupt also needs to be set.

IOCTL_ORB2_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (unsigned long integer)
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel interrupts. The interrupts that deal with the DMA transfers do not affect this value. Masked version of channel status.

IOCTL_ORB2_CHAN_SWW_TX_FIFO
Function: Writes a 32-bit data word to the transmit FIFO.
Input: FIFO word (unsigned long integer)
Output: none
Notes: Used to make single-word accesses to the transmit FIFO instead of using DMA.

IOCTL_ORB2_CHAN_SWR_RX_FIFO
Function: Returns a 32-bit data word from the receive FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

IOCTL_ORB2_CHAN_SET_CONT
Function: write to Channel Control register using structure
Input: ORB2_CHAN_CONT
Output: None
Notes: See DDORB2Chan.h for structure. See below for quick reference.

IOCTL_ORB2_CHAN_GET_CONT
Function: Read from Channel Control register using structure
Input: None
Output: ORB2_CHAN_CONT
Notes: See DDORB2Chan.h for structure. See below for quick reference.

```
FifoTestEn;   // BiPass Mode Control
MIntEn;       // Master Interrupt Enable
WrDmaEn;      // Write DMA Interrupt Enable
RdDmaEn;      // Read DMA Interrupt Enable
TxUrgent;     // Set to give the TX DMA on this  channel a higher priority
RxUrgent;     // Set to give the RX DMA on this channel a higher priority
```

GP registers are used for a variety of purposes depending on the COM port. For Ternary the registers are used to store the register data transmitted with StartTxReg. For LS and HS the sync pattern is stored. For TLM register files take the place of these registers. Please refer to the HW manual for more details.

IOCTL_ORB2_CHAN_SET_TX_REG
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TXREG
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TX_REG
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TXREG
Notes: See DDORB2Chan.h for structure.

TX_COUNT is used to set the delay and the size of the data sent per packet.

IOCTL_ORB2_CHAN_SET_TX_COUNT
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TXCOUNT
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TX_COUNT
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TXCOUNT
Notes: See DDORB2Chan.h for structure.

The transmit control register has the same address for each COM port and different bit assignments or meanings. Each COM port has a separate TX and RX SET and GET command to access the particular port with a unique structure for that port. Please be careful to use the correct channel number when accessing these ports.

Ternary Control

IOCTL_ORB2_CHAN_SET_TX_COM12
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TXCOM12_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TX_COM12
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TXCOM12_CONTROL
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_RX_COM12
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_RXCOM12_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_RX_COM12
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_RXCOM12_CONTROL
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_SYNC_COM12
Function: write to Channel Sync registers using structure
Input: ORB2_CHAN_SYNCREG12
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_SYNC_COM12
Function: write to Channel Sync registers using structure
Input: None
Output: ORB2_CHAN_SYNCREG12
Notes: See DDORB2Chan.h for structure.

LS Control

IOCTL_ORB2_CHAN_SET_TX_COM34
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TXCOM34_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TX_COM34
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TXCOM34_CONTROL
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_RX_COM34
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_RXCOM34_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_RX_COM34
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_RXCOM34_CONTROL
Notes: See DDORB2Chan.h for structure.

TLM Control

IOCTL_ORB2_CHAN_SET_MAS_COM56
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_MASCOM56_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_MAS_COM56
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_MASCOM56_CONTROL
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_TAR_COM56
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TARCOM56_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TAR_COM56
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TARCOM56_CONTROL
Notes: See DDORB2Chan.h for structure.

TLM Register Files

IOCTL_ORB2_CHAN_SET_MAS_RF_COM56
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_MASREG
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_MAS_RF_COM56
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_MASREG
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_TAR_RF_COM56
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TARREG

Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TAR_RF_COM56
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TARREG
Notes: See DDORB2Chan.h for structure.

HS Control

IOCTL_ORB2_CHAN_SET_TX_COM78
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_TXCOM78_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_TX_COM78
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_TXCOM78_CONTROL
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_SET_RX_COM78
Function: write to Channel GP registers using structure
Input: ORB2_CHAN_RXCOM78_CONTROL
Output: None
Notes: See DDORB2Chan.h for structure.

IOCTL_ORB2_CHAN_GET_RX_COM78
Function: Read from Channel GP registers using structure
Input: None
Output: ORB2_CHAN_RXCOM78_CONTROL
Notes: See DDORB2Chan.h for structure.

Length Error Counts

IOCTL_ORB2_CHAN_SET_RX_LEN_ERR_CNT
Function: write to Channel Length Error Counter – Preload function
Input: ULONG
Output: None
Notes: The counter can be “preloaded” to 0 to clear or to some other value as needed for your system. The count is incremented when a length error is detected . Only applies to packet modes. Ternary, LS and HS [VP mode].

IOCTL_ORB2_CHAN_GET_RX_LEN_ERR_CNT
Function: Read from Channel Length Error Counter
Input: None
Output: ULONG
Notes: Read back preloaded value plus increment count. Count advances once per Long [too much data] or short [not enough data] error occurrences in modes where the length is checked – “packetized”. The type of status can be appended to the data [see RX control structure for the port in use].

Write

DMA data is written to the referenced I/O channel device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Read

DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchandisability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be “cockpit error” rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer’s making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer’s invoicing policy.

Out of Warranty Repairs

Out of warranty support will be billed. The current minimum repair charge is $125. An open PO will be required.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax

support@dyneng.com

Appendix

Reference copy of structures for evaluation

The following structures shown are available in the DDORBChan.h and DDORB2Base.h files included with the driver. The structures are included here for your evaluation when considering the driver package. The electronic versions included with the driver should be used with your project. The names track the register bit definitions. For details about particular signals please refer to the HW manual.

Base:

```
#define PLL_MESSAGE1_SIZE       16
#define PLL_MESSAGE2_SIZE       24
#define PLL_MESSAGE_SIZE        (PLL_MESSAGE1_SIZE + PLL_MESSAGE2_SIZE)

 // Driver/Device information
typedef struct _ORB2_BASE_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   UCHAR    XilinxVersion;
   UCHAR    XilinxDesign;
   UCHAR    PllDeviceId;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
} ORB2_BASE_DRIVER_DEVICE_INFO, *PORB2_BASE_DRIVER_DEVICE_INFO;

typedef struct _ORB2_BASE_PLL_DATA
{
   UCHAR    Data[PLL_MESSAGE_SIZE];
} ORB2_BASE_PLL_DATA, *PORB2_BASE_PLL_DATA;
```

Channel:

```
typedef enum _ORB2_CHAN_MODE_SEL {ORB2_PV, ORB2_VO, ORB2_SO, ORB2_CO}
ORB2_CHAN_MODE_SEL, *PORB2_CHAN_MODE_SEL;

typedef struct _ORB2_CHAN_DRIVER_DEVICE_INFO
{
  UCHAR    DriverVersion;
  ULONG    InstanceNumber;
} ORB2_CHAN_DRIVER_DEVICE_INFO, *PORB2_CHAN_DRIVER_DEVICE_INFO;

typedef enum _ORB2_CHAN_FIFO_SEL {ORB2_TX, ORB2_RX, ORB2_EXT, ORB2_ALL,
ORB2_BOTH} ORB2_CHAN_FIFO_SEL, *PORB2_CHAN_FIFO_SEL;

typedef struct _ORB2_CHAN_FIFO_LEVELS
{
  USHORT   AlmostFull;
  USHORT   AlmostEmpty;
} ORB2_CHAN_FIFO_LEVELS, *PORB2_CHAN_FIFO_LEVELS;

typedef struct _ORB2_CHAN_FIFO_COUNTS
{
  USHORT   RxCountwPipe;
  USHORT   TxCount;
} ORB2_CHAN_FIFO_COUNTS, *PORB2_CHAN_FIFO_COUNTS;

typedef struct _ORB2_CHAN_CONT
{
  BOOLEAN                    FifoTestEn;// BiPass Mode Control
  BOOLEAN                    MIntEn;    // Master Interrupt Enable
  BOOLEAN                    WrDmaEn;   // Write DMA Interrupt Enable
  BOOLEAN                    RdDmaEn;   // Read DMA Interrupt Enable
  BOOLEAN                    TxUrgent;  // Enable Higher Priority Processing for TX
  BOOLEAN                    RxUrgent;  // Enable Higher Priority Processing for RX
} ORB2_CHAN_CONT, *PORB2_CHAN_CONT;
```

```
typedef struct _ORB2_CHAN_TXCOM12_CONTROL
{
  BOOLEAN          TxStartReg; // start transmit state machine or stop, using Register based
data can be auto cleared
  BOOLEAN          TxStartDma;   // start transmit state machine or stop using DMA data
  BOOLEAN          TxSmIntEn;    // Set to enable Transmit Interrupt from State Machine
  BOOLEAN          TxFifoIntEn;  // Set to enable Transmit FIFO based Interrupt
  BOOLEAN          TxDataOrder;  // Set to reverse, clear for pass through
  BOOLEAN          TxSMode;      // Set for S-1, cleared for S-0 encoding
  BOOLEAN          TxSInv;       // Set for inverted, cleared for standard S encoding
  BOOLEAN          TxClkPol;     // Set for Falling Edge Valid, Clear for Rising Edge Valid for
Transmitted data
  UCHAR            TxClkDiv;     // select clock frequency 8 bits [PLLA is reference, divide
by 2[n+1]
} ORB2_CHAN_TXCOM12_CONTROL, *PORB2_CHAN_TXCOM12_CONTROL;

typedef struct _ORB2_CHAN_RXCOM12_CONTROL
{
  BOOLEAN          RxStart;              // start or stop Receive state machine
  BOOLEAN          RxStatusDisable; // Set to disable Status from being appended to Packet
data stored in FIFO
  BOOLEAN          RxSmIntEn;            // Set to enable Receive State Machine Interrupt
  BOOLEAN          RxFifoIntEn;          // Set to enable Receive FIFO based Interrupt
  BOOLEAN          RxOverflowIntEn;      // Set to enable FIFO Overflow based Interrupt
  BOOLEAN          RxSMode;              // Set for S-1, cleared for S-0 encoding
  BOOLEAN          RxSInv;               // Set for inverted, cleared for standard S encoding
  BOOLEAN          RxDataOrder;          // Set to reverse, clear for pass through
  BOOLEAN          RxClkPol;     // Set for Falling Edge Valid, Clear for Rising Edge Valid for
Received data
  BOOLEAN          RxDataFill;   // Set for 1's, Clear for 0's padding for missing or shifted
down data on non aligned words
} ORB2_CHAN_RXCOM12_CONTROL, *PORB2_CHAN_RXCOM12_CONTROL;

typedef struct _ORB2_CHAN_TXREG
{
  ULONG                          TxReg0;  // Data Register 0 - First to transmit from
  ULONG                          TxReg1;  // Data Register 1 -
  ULONG                          TxReg2;  // Data Register 2 -
  ULONG                          TxReg3;  // Data Register 3 -
  ULONG                          TxReg4;  // Data Register 4 -
  ULONG                          TxReg5;  // Data Register 5 -
  ULONG                          TxReg6;  // Data Register 6 -
  ULONG                          TxReg7;  // Data Register 7 - Last to transmit from
} ORB2_CHAN_TXREG, *PORB2_CHAN_TXREG;
```

```
typedef struct _ORB2_CHAN_TXCOUNT
{
  USHORT     TxBitCount;  // Bit Count to Transmit, Word count in COM3,4 N+1 sent
  USHORT     TxToneCount; // Minimum Delays [tone mode] between packet transmissions
N>=3 N+1 delayed
} ORB2_CHAN_TXCOUNT, *PORB2_CHAN_TXCOUNT;

typedef struct _ORB2_CHAN_TXCOM34_CONTROL
{
  BOOLEAN            TxStartDma;   // start transmit state machine or stop using DMA data
  BOOLEAN            TxSmIntEn;    // Set to enable Transmit Interrupt from State Machine
  BOOLEAN            TxFifoIntEn;  // Set to enable Transmit FIFO based Interrupt
  BOOLEAN            TxDataOrder;  // Set to reverse, clear for pass through
  BOOLEAN            TxMode;       // Set for packetized, Clear for FIFO based transmission
  BOOLEAN            TxClkPol;     // Clear for standard Falling edge valid data, set for rising
edge valid
  BOOLEAN            TxClkMaskEn; // Set for clock active during data only, clear for free
running
  UCHAR              TxClkDiv;     // select clock frequency 8 bits [PLLA is reference, divide
by 2[n+1]
} ORB2_CHAN_TXCOM34_CONTROL, *PORB2_CHAN_TXCOM34_CONTROL;

typedef struct _ORB2_CHAN_RXCOM34_CONTROL
{
  BOOLEAN            RxStart;              // start or stop Receive state machine
  BOOLEAN            RxStatusDisable;      // Set to disable Status being added to Packet data
  BOOLEAN            RxSmIntEn;            // Set to enable Receive State Machine Interrupt
  BOOLEAN            RxFifoIntEn;          // Set to enable Receive FIFO based Interrupt
  BOOLEAN            RxOverflowIntEn;      // Set to enable FIFO Overflow based Interrupt
  BOOLEAN            RxValidMode;          // set for Valid, clear for clock decoding
  BOOLEAN            RxDataOrder;          // Set to reverse, clear for pass through
  BOOLEAN            RxClkPol;             // Clr (std) Falling edge valid, set rising edge valid
  BOOLEAN            RxDataFill;           // Set for 1's, Clear for 0's padding for missing or
shifted down data on non aligned words
  ORB2_CHAN_MODE_SEL RxModeSm;    // ORB2_PV [Packet Valid], ORB2_VO [Valid Only],
ORB2_SO [Sync Only], ORB2_CO [Clk Only]
  BOOLEAN            RxSyncEn0;            // Set to include 7-0 in synch check
  BOOLEAN            RxSyncEn1;            // Set to include 15-8 in synch check
  BOOLEAN            RxSyncEn2;            // Set to include 23-16 in synch check
  BOOLEAN            RxSyncEn3;            // Set to include 31-24 in synch check
  BOOLEAN            RxSyncEn4;            // Set to include 39-32 in synch check
  BOOLEAN            RxSyncEn5;            // Set to include 47-40 in synch check
  BOOLEAN            RxSyncEn6;            // Set to include 55-48 in synch check
  BOOLEAN            RxSyncEn7;            // Set to include 63-56 in synch check
  BOOLEAN            RxSyncSearch;         // set to cause mode 10 to reacquire the sync
pattern, auto cleared when pattern detected

} ORB2_CHAN_RXCOM34_CONTROL, *PORB2_CHAN_RXCOM34_CONTROL;
```

```
typedef struct _ORB2_CHAN_MASCOM56_CONTROL
{
  BOOLEAN          MStartReg;     // start transmit state machine
  BOOLEAN          MIoEn;         // Enable IO for Master Fucntion [Clk and Gate]
  BOOLEAN          MSmIntEn;      // Set to enable Transmit Interrupt from State Machine
  BOOLEAN          MClkPol;       // Clear for standard Falling edge valid data, set for rising
edge valid
  BOOLEAN          MClkMaskEn;  // Set for clock active during data only, clear for free
running
  BOOLEAN          MGatePolarity;          // Set for active low, Clear for active high Gate
  BOOLEAN          MSysSmDataIn;       // Set for test path data, clear for normal operation
=> input to Master Register File
  BOOLEAN          MTImType;               // Clear for Type 1 Set for Type II
  UCHAR            MClkDiv;                // select clock frequency 8 bits [PLLA is reference,
divide by 2[n+1]
} ORB2_CHAN_MASCOM56_CONTROL, *PORB2_CHAN_MASCOM56_CONTROL;

typedef struct _ORB2_CHAN_TARCOM56_CONTROL
{
  BOOLEAN          TStart;                 // start or stop Receive state machine
  BOOLEAN          TSmIntEn;               // Set to enable Receive State Machine Interrupt
  BOOLEAN          TClkPol;                // Clr (std)Falling edge valid, set rising edge valid
  BOOLEAN          TGatePolarity;          // Set for active low, Clear for active high Gate
  BOOLEAN          TTImType;               // Clear for Type 1 Set for Type II

} ORB2_CHAN_TARCOM56_CONTROL, *PORB2_CHAN_TARCOM56_CONTROL;


//  Master Mode Register File
//    When writing be sure to use test mode setting in Master Control register for data path
//      Data stored here when master mode is tasked to retrieve external data
typedef struct _ORB2_CHAN_MASREG
{
  ULONG                            MReg0;  // Data Register 0 - First register read in
  ULONG                            MReg1;  // Data Register 1 -
  ULONG                            MReg2;  // Data Register 2 -
  ULONG                            MReg3;  // Data Register 3 -
  ULONG                            MReg4;  // Data Register 4 -
  ULONG                            MReg5;  // Data Register 5 -
  ULONG                            MReg6;  // Data Register 6 -
  ULONG                            MReg7;  // Data Register 7 - Last read in
} ORB2_CHAN_MASREG, *PORB2_CHAN_MASREG;
```

```
// Targat Mode Register File
//   Store data here to transmit out when tasked from external device
typedef struct _ORB2_CHAN_TARREG
{
  ULONG                                TReg0;  // Data Register 0 - First to transmit from
  ULONG                                TReg1;  // Data Register 1 -
  ULONG                                TReg2;  // Data Register 2 -
  ULONG                                TReg3;  // Data Register 3 -
  ULONG                                TReg4;  // Data Register 4 -
  ULONG                                TReg5;  // Data Register 5 -
  ULONG                                TReg6;  // Data Register 6 -
  ULONG                                TReg7;  // Data Register 7 - Last to transmit from
} ORB2_CHAN_TARREG, *PORB2_CHAN_TARREG;

typedef struct _ORB2_CHAN_TXCOM78_CONTROL
{
  BOOLEAN          TxStartDma;    // start transmit state machine or stop using DMA data
  BOOLEAN          TxSmIntEn;     // Set to enable Transmit Interrupt from State Machine
  BOOLEAN          TxFifoIntEn;   // Set to enable Transmit FIFO based Interrupt
  BOOLEAN          TxDataOrder;   // Set to reverse, clear for pass through
  BOOLEAN          TxMode;        // Set for packetized, Clear for FIFO based transmission
  BOOLEAN          TxClkPol;      // Clr (std) Falling edge valid, set rising edge valid
  BOOLEAN          TxClkMaskEn; // Set for bursted, clear for free running
  BOOLEAN          TxDataValidPol; // Set for inversion, clear for standard operation
  BOOLEAN          TxFifoMuxCont;         // clear to select Tx Path into external FIFO, Set to
select Rx path to external FIFO
  BOOLEAN          TxFifoLoad;    // Set or Clear to control access to Programmable registers

} ORB2_CHAN_TXCOM78_CONTROL, *PORB2_CHAN_TXCOM78_CONTROL;
```

```
typedef struct _ORB2_CHAN_RXCOM78_CONTROL
{
   BOOLEAN              RxStart;                // start or stop Receive state machine
   BOOLEAN              RxStatusDisable;        // Set to disable Status being appended to Packet
   BOOLEAN              RxSmIntEn;              // Set to enable Receive State Machine Interrupt
   BOOLEAN              RxFifoIntEn;            // Set to enable Receive FIFO based Interrupt
   BOOLEAN              RxOverflowIntEn;        // Set to enable FIFO Overflow based Interrupt
   BOOLEAN              RxValidMode;            // set for Valid, clear for clock decoding
   BOOLEAN              RxDataOrder;            // Set to reverse, clear for pass through
   BOOLEAN              RxClkPol;       // Clr (std) Falling edge valid set for rising edge valid
   BOOLEAN              RxDataFill;             // Set for 1's, Clear for 0's padding for missing or
shifted down data on non aligned words
   BOOLEAN              RxDataValidPol;         // Set for inversion, clear for standard operation
   BOOLEAN              RxFifoPathEn;           // Clear to disable, Set to enable auto read from
external FIFO into RX DMA FIFO
   ORB2_CHAN_MODE_SEL RxModeSm;    // ORB2_PV [Packetized, Valid], ORB2_VO [Valid
Only], ORB2_SO [Sync Only], ORB2_CO [Clk Only]
   BOOLEAN              RxSyncEn0;                      // Set to include 7-0 in synch check
   BOOLEAN              RxSyncEn1;                      // Set to include 15-8 in synch check
   BOOLEAN              RxSyncEn2;                      // Set to include 23-16 in synch check
   BOOLEAN              RxSyncEn3;                      // Set to include 31-24 in synch check
   BOOLEAN              RxSyncEn4;                      // Set to include 39-32 in synch check
   BOOLEAN              RxSyncEn5;                      // Set to include 47-40 in synch check
   BOOLEAN              RxSyncEn6;                      // Set to include 55-48 in synch check
   BOOLEAN              RxSyncEn7;                      // Set to include 63-56 in synch check
   BOOLEAN              RxSyncSearch;                   // set to cause mode 10 to reacquire the
sync pattern, auto cleared when pattern detected

} ORB2_CHAN_RXCOM78_CONTROL, *PORB2_CHAN_RXCOM78_CONTROL;
```